

FPGA implementation of a 32x32 autocorrelator array for analysis of fast image series

Jan Buchholz,¹ Jan Wolfgang Krieger,¹ Gábor Mocsár,² Balázs Kreith,² Edoardo Charbon,³ György Vámosi,² Udo Kebschull,⁴ and Jörg Langowski^{1, a)}

¹⁾ German Cancer Research Center (DKFZ), Biophysics of Macromolecules (B040), Im Neuenheimer Feld 580, D-69120 Heidelberg, Germany

²⁾ University of Debrecen, Medical and Health Science Center, Faculty of Medicine, Department of Biophysics and Cell Biology, H-4032 Debrecen, Nagyerdei krt. 98, Hungary

³⁾ Technische Universiteit Delft, Mekelweg 4, 2628 CD Delft, The Netherlands

⁴⁾ Goethe-Universität Frankfurt, Senckenberganlage 31, D-60325 Frankfurt, Germany

(Dated: 8 December 2011)

With the evolving technology in CMOS integration, new classes of 2D-imaging detectors have recently become available. In particular, single photon avalanche diode (SPAD) arrays allow detection of single photons at high acquisition rates (≥ 100 kfps), which is about two orders of magnitude higher than with currently available cameras. Here we demonstrate the use of a SPAD array for imaging fluorescence correlation spectroscopy (imFCS), a tool to create 2D maps of the dynamics of fluorescent molecules inside living cells. Time-dependent fluorescence fluctuations, due to fluorophores entering and leaving the observed pixels, are evaluated by means of autocorrelation analysis. The multi- τ correlation algorithm is an appropriate choice, as it does not rely on the full data set to be held in memory. Thus, this algorithm can be efficiently implemented in custom logic. We describe a new implementation for massively parallel multi- τ correlation hardware. Our current implementation can calculate 1024 correlation functions at a resolution of $10\ \mu\text{s}$ in real-time and therefore correlate real-time image streams from high speed single photon cameras with thousands of pixels.

PACS numbers: 42.50.Ar, 42.62.Fi, 78.47.je

Keywords: Correlator; FPGA; SPIM; SPAD; FCS; Photon counting and statistics

I. INTRODUCTION

Fluorescence correlation spectroscopy^{1,2} (FCS) is a powerful experimental technique for measuring the dynamics of fluorescently labeled molecules in solution and also inside living cells. It allows one to determine the particle number, the diffusion coefficient, flow speeds and also photophysical and chemical reaction rates (for an overview, see Ref. 3). In FCS the time trace of the fluorescence intensity fluctuations $I(t)$ inside a small observation volume (usually around $10^{-15}\text{ l} = 1\ \mu\text{m}^3$) is monitored. The fluctuations originate from particles entering and leaving the focus, or transitions between states having different quantum yields. Faster dynamics of the fluorescing particles also lead to faster fluctuations, which can be quantified by means of a temporal first-order autocorrelation function (ACF):

$$g(\tau) = \frac{\langle I(t) \cdot I(t + \tau) \rangle_t}{\langle I(t) \rangle_t^2}, \quad \langle I(t) \rangle_t := \lim_{\tilde{T} \rightarrow \infty} \frac{1}{\tilde{T}} \int_0^{\tilde{T}} I(t) \, dt \quad (1)$$

The ACF usually contains features that are spread over several orders of magnitude in time (nanoseconds to seconds).

The standard FCS setup uses a confocal microscope in combination with single-photon sensitive detectors to

acquire the fluorescence time trace $I(t)$ from one focal volume. Then the data are fed into a “correlator” (hardware or software component), which estimates the ACF over a certain dynamic range.

In the accompanying paper by Mocsár *et al.*⁴, a field programmable gate array (FPGA) implementation of such a correlator circuit for use with a confocal microscope and up to two single photon avalanche diode (SPAD) detectors is presented. In recent years, the availability of fast cameras has triggered the development of different imaging modalities for FCS^{5–7}, which allow spatial mapping of the diffusion coefficient and other dynamic properties by calculating an ACF for each of potentially many pixels. This requires correlation hardware that can process the data stream very fast, ideally in real time, for all pixels of the image sensor. Here we extend the idea of hardware reuse, presented before⁴, for application to imaging FCS.

Our FPGA-based implementation can calculate 1024 autocorrelation functions in parallel and in real time. The dynamic range of the ACFs is $\tau_{\min} \dots \tau_{\max} = 10\ \mu\text{s} \dots 1\text{ s}$. As a data acquisition device we use the single photon avalanche diode array (SPAD array) *Radhard2*^{8,9}. From our sensor we read a 32×32 pixel frame every $\Delta t_{\text{frame}} = 10\ \mu\text{s}$ where each pixel contains 1 bit of information (no photon or at least one photon in the last Δt_{frame}). The design presented here could easily be adapted to other image sensors such as customized complementary metal oxide semiconductor (CMOS) or electron multiplying charge-coupled device (EMCCD) cam-

^{a)}e-mail: jl@dkfz.de; homepage: <http://www.dkfz.de/Macromol>

eras.

II. MULTI- τ HARDWARE CORRELATORS

A hardware correlator estimates the ACF in Eq. 1 from a finite sequence of intensity measurements

$$I_n = \int_0^{\tau_{\min}} I(n \cdot \tau_{\min} + t) dt, \quad n = 0, 1, \dots, T - 1 \quad (2)$$

with the number of samples T and the integration time τ_{\min} for one sample. When discretizing Eq. 1 with this intensity sequence, care has to be taken not to bias the normalization $1/\langle I \rangle_t^2$. A viable choice is the “symmetric normalization” introduced in Ref. 10:

$$\hat{g}_{\text{sym}}(\tau_k) = \frac{\frac{1}{T-\tau_k} \cdot \overbrace{\sum_{n=\tau_k}^{T-1} I_n \cdot I_{n-\tau_k}}^{=:G_{\tau_k}}}{\left[\underbrace{\frac{1}{T} \cdot \sum_{n=0}^{T-1} I_n}_{=:M_0} \right] \cdot \left[\underbrace{\frac{1}{T-\tau_k} \cdot \sum_{n=\tau_k}^{T-1} I_{n-\tau_k}}_{=:M_{\tau_k}} \right]} \quad (3)$$

with a given set of lag times $\tau_k \in \mathbb{N}$ (in units of τ_{\min} , so $\tau = \tau_k \cdot \tau_{\min}$). When the full sequence $\{I_n\}_{n=0\dots T-1}$ is available after the measurement, Eq. 3 may be evaluated directly for an arbitrary (also logarithmically spaced) set of lags τ_k . This gives an unbiased estimation of the ACF (“direct correlation”).

To implement our hardware correlator, we use the multi- τ scheme introduced in Ref. 11, which is also illustrated and compared to a linear implementation in FIG. 1 (for a detailed description, please refer to our accompanying paper Ref. 4). The multi- τ scheme uses a set of S “linear” correlator blocks (FIG. 1(b,c)). The input samples $I_{s,n}$ (n is the same index as in Eq. 2) are summed over increasingly long periods $\Delta n = m^s$:

$$I_{s,n} = \sum_{k=1}^{m^s} I_{n-k}, \quad \text{for } s > 0 \quad (4)$$

with $I_{0,n} = I_n$.

Each of the linear correlators estimates the ACF at P linearly spaced lags

$$\begin{aligned} \tau_{0,0} &= 0 \\ \tau_{s,0} &= \tau_{s-1,P-1} + m^{s-1} \\ \tau_{s,p} &= \tau_{s,p-1} + m^s = \sum_{i=1}^{s \cdot P + p} m^{\lfloor \frac{i-1}{P} \rfloor}, \end{aligned} \quad (5)$$

where $p = 0 \dots P - 1$.

In summary, this results in a quasi-logarithmic spacing of estimates $\hat{g}_{\text{sym,multi-}\tau}(\tau_{s,p})$. The advantage of this multi- τ scheme is its simple implementation in hardware

and a large dynamic time range with a reasonable number of channels. Its disadvantage is a systematic error introduced by averaging: As shown in Ref. 12 the estimator $\hat{g}_{\text{sym,multi-}\tau}(\tau_{s,p})$ equals the ideal correlation function $g(\tau_{s,p} \cdot \tau_{\min})$ (see Eq. 3) convolved with a triangular kernel with width m^s :

$$\hat{g}_{\text{sym,multi-}\tau}(\tau_{s,p}) = g(\tau_{s,p} \cdot \tau_{\min}) * \Lambda(\tau_{s,p}, m^s), \quad (6)$$

where $*$ denotes the convolution product and $\Lambda(\tau, \Delta\tau) = \Delta\tau - |\tau|$ for $|\tau| < \Delta\tau$ and $\Lambda(\tau, \Delta\tau) = 0$ for $|\tau| \geq \Delta\tau$, is the triangular shaped kernel.

III. HARDWARE DESIGN

Here we describe how the hardware reuse scheme in the accompanying paper Ref. 4 can be extended to accommodate many more input channels. Instead of the graphical tool used there, here we employed a low-level hardware description language to gain speed and flexibility. This enables us to fine-tune many parameters of the final design, e.g. operational speed, memory usage, logic resource consumption and routing between logic cells.

A. Single-Pixel Correlator

As shown in FIG. 1, a typical correlator is made up from channels, each corresponding to a certain lag time and consisting of a multiplier, an accumulator and a delay element.

The idea of our implementation is to use one single channel circuit to calculate all channels within one multi- τ correlator. This is possible by serial processing of the lag time channels, since every channel’s hardware is identical.

The basic arithmetic operation of one channel is to multiply and accumulate (MAC). Therefore we can map its functionality onto a MAC unit which can be found on most FPGA architectures and which is considerably faster than using generic FPGA logic cells. Only about 100 of these can be found on typical FPGAs, precluding any approach with blocks consisting of several lag channels.

As only one circuit is used to process all channels, we use an internal memory block (block random access memory, BRAM) to store their state (i.e., the content of the accumulator and the delayed signal). We implement this circuit by decomposing it into five steps:

1. **Load** accumulator and delayed value of a channel from memory
2. **Wait** for memory access to complete
3. **Multiply** delayed with global signal
4. **Add** multiplication result to channel’s accumulator

5. Store counter and new delayed value to memory

To increase performance, these steps are executed in an interleaved manner for four channels simultaneously. This “pipelining” scheme is shown in TAB. I. Our five pipeline steps are compatible with the internal pipelining of common MAC units.

TABLE I. Interleaved pipeline of the linear correlator design with 8 channels.

cycle c	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	
ch. 0	L W M A S	L W M A S	ch. 4
ch. 1	L W M A S	L W M A S	ch. 5
ch. 2	L W M A S	L W M A S	ch. 6
ch. 3	L W M A S	L W M A S	ch. 7

From one block to the next the delay time is doubled ($m = 2$) in the multi- τ scheme, making the input data rate of block $s + 1$ half that of block s . Hence we need to execute each block only half as often as its predecessor. Thus, a complete multi- τ correlator can be executed in only twice the run-time Δt_{lin} of a single linear correlator block:

$$\begin{aligned} & \underbrace{1 \cdot \Delta t_{\text{lin}}}_{1^{\text{st}} \text{ lin. corr.}} + \underbrace{\frac{1}{2} \cdot \Delta t_{\text{lin}}}_{2^{\text{nd}} \text{ lin. corr.}} + \underbrace{\frac{1}{4} \cdot \Delta t_{\text{lin}}}_{3^{\text{rd}} \text{ lin. corr.}} + \dots \\ & \leq \sum_{n=0}^{\infty} \frac{1}{2^n} \cdot \Delta t_{\text{lin}} = 2 \cdot \Delta t_{\text{lin}} \quad (7) \end{aligned}$$

A key requirement is that Δt_{lin} is at most half the integration time τ_{\min} of the input signal I_n .

As before⁴, a scheduler guarantees that a linear correlator block s is only executed after its predecessor $s - 1$ has been executed twice. A counter $c = 0, 1, \dots$ is incremented with every execution of any linear correlator block. The scheduler uses the following relations to determine which linear correlator block s has to be executed at a given counter value c (details see appendix):

$$\begin{aligned} s = 0 : & \quad c \bmod 2^1 = 0 \\ s = 1 : & \quad c \bmod 2^2 = 3 \\ s \geq 2 : & \quad c \bmod 2^{s+1} = (2^s - 3) \end{aligned} \quad (8)$$

FIG. 2(a) shows the solution of this relation for c values from 0 to 31. In the binary representation of c for a linear correlator block s patterns are evident that can be used to implement the scheduler efficiently. As shown in TAB. II (for $S = 8$ linear correlators), correlator block $s = 0$ is executed whenever the last bit of c is 0_b , correlator $s = 1$ is executed when the last two bits are 11_b and so forth. This scheme uses only simple comparison operations.

Between two consecutive blocks $s - 1$ and s , adder circuitry is inserted to sum up two subsequent input signal values $I_{s-1,n-1}$ and $I_{s-1,n}$. This is done for both the delayed/local as well as the undelayed/global signal, while they are processed in the pipeline.

TABLE II. Binary representation of counter c that solves relations Eq. 8 for a given linear correlator block s . * denotes a don’t care condition.

lin. corr. s	binary representation of counter c that solves relation Eq. 8								
	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	*	*	*	*	*	*	*	*	0
1	*	*	*	*	*	*	*	1	1
2	*	*	*	*	*	*	0	0	1
3	*	*	*	*	*	0	1	0	1
4	*	*	*	*	0	1	1	1	0
5	*	*	*	0	1	1	1	0	1
6	*	*	0	1	1	1	1	0	1
7	*	0	1	1	1	1	1	0	1

The linear correlator as implemented here, together with its scheduler and the summation logic, is called correlation processing element (CorrPE). All channel data and intermediate summation results, the so-called pixel context, are stored in a dual-port BRAM which is associated with the CorrPE.

B. Multi-Pixel Correlator

The CorrPE described above is much faster than required to calculate the ACF for a single pixel in the SPAD array. Hence, we can reuse a single CorrPE for multiple pixels by switching between pixel contexts. This “pixel scheduler” uses a double-buffering strategy. While a CorrPE operates on the current context, the previous context is exchanged with the next context to be processed. The pixel contexts are stored in external background memory (SRAM), because they exceed the capacity of the internal memory. In FIG. 3 we illustrate how we reuse one CorrPE for several pixels in comparison to a naive implementation using one CorrPE per pixel.

In addition to the channel data, the cycle counter c and an accumulator for the local and the global input signals have to be saved. The latter are used for normalization and cross-correlation.

A single CorrPE is used to process an entire column (ACFs of $n_y = 32$ pixels) of our SPAD array. To handle the full $n_x \times n_y$ array of pixels, we instantiate $n_x = 32$ CorrPEs in parallel. An overview of this scheme is shown in FIG. 4. A “data acquisition” circuit communicates with the SPAD array and provides the image data for the correlators. As the image data are streamed out row by row, and each of the n_x CorrPEs is only processing data from one specific context (i.e. a specific row), the remaining pixels have to be buffered in 32 FIFOs (first-in first-out memory buffer) localized in external RAM.

In addition our design contains two USB 2.0 interfaces. One is used to send the raw data stream from the SPAD array to the computer, which allows further data processing. We also use these raw data to verify our correlator design. Via the second interface, intermediate and final

results from the correlators are transferred to the host computer. The intermediate results allow implementing a live view of the ongoing calculation of the ACFs.

C. ACF Normalization

The intensity values in the denominator of Eq. 3 are typically obtained from *monitor channels* M_{τ_k} , which accumulate the total photon count at a given lag time τ_k . In contrast to other implementations, we use only one monitor M_0 (input signal accumulator) per multi- τ correlator. Symmetric normalization (see also Eq. 3) yields the following:

$$\hat{g}_{\text{sym,multi-}\tau}(\tau_{s,p}) = \frac{G_{\tau_{s,p}}}{2^s} \cdot \frac{T}{M_0 \cdot M_{\tau_{s,p}}} \quad (9)$$

After measuring T samples, the number of samples that have propagated through the correlator to a distinct channel (s, p) is $T - \tau_{s,p}$. Under the assumption that I_n is a stationary random process and that $T > \tau_{s,p}$, the per-channel monitors $M_{\tau_{s,p}}$ can be estimated in the following manner:

$$M_{\tau_{s,p}} = M_0 \cdot \frac{T - \tau_{s,p}}{T}. \quad (10)$$

This normalization and model fitting is done on the host computer, since floating-point arithmetic cannot be implemented efficiently on an FPGA. Although the correlation on the FPGA and the normalization on the host computer can easily be done in real time, the fits in most cases cannot. Usually a single curve fit takes tens of milliseconds, thus fitting all 1024 ACFs would amount to ≥ 10 s additional computing time; but this still allows to display fit parameter maps (images) within an acceptably short delay. In current imaging FCS software systems¹³ the data are loaded and correlated on the host computer, which for a measurement of typically 10 s takes ≥ 10 min for 1024 pixels at the frame rate of our sensor.

To show that the estimation in Eq. 10 yields good results, we simulated different correlator types in software. The results for a direct estimation of the ACF using Eq. 3 (green), a multi- τ correlator with a monitor channel per lag (blue) and our estimation (magenta) can be seen in FIG. 5, where the data in (a) and (b) were obtained by correlating the input signal $I(t) = 1 + \sin(2\pi t/(1.51 \cdot 10^{-4}))$ for which the exact ACF is known to be $g^{(\text{theoretical})}(\tau) = 1 + \cos(2\pi\tau/(1.51 \cdot 10^{-4}))$ (time t and lags τ are unit free). The data in FIG. 5(c) was created by simulating a $T_{\text{sim}} = 1$ s long FCS experiment with one diffusing species¹⁴. It was computed with our FCS simulation software described in Ref. 15. Further details on the simulation code are shown in the appendix.

For short lags the estimated ACFs resemble the theoretical curves quite well. Multi- τ correlators have an

increased absolute error for longer lags, which is due to the averaging described in Eq. 6. This can be seen especially in the case of the sine wave signal. The multi- τ estimates can still be used for FCS experiments, as here the ACFs usually decay to 1 (white noise) for large lag times, and thus the systematic error drops to zero again (for a detailed discussion of this, see e.g. Ref. 12). For $\tau \gtrsim T_{\text{sim}}/10$, multi- τ correlators show additional systematic deviations from the theoretical curve and from the direct estimation, because the channels are not averaged over sufficiently many samples to yield reliable results. Here the multi- τ implementation with multiple monitor channels performs better due to the better estimation of the normalization factor $M_{\tau_{s,p}}$.

D. Crosscorrelation (CCF)

Our design can also calculate cross-correlation:

$$g^{(x,y)}(\tau) = \frac{\langle I^{(x)}(t) \cdot I^{(y)}(t + \tau) \rangle_t}{\langle I^{(x)}(t) \rangle_t \cdot \langle I^{(y)}(t) \rangle_t}$$

between two input signals $I^{(x)}(t)$ and $I^{(y)}(t)$ at the local $J^{(l)}$ and global $J^{(g)}$ inputs of the CorrPE (see FIG. 1 where both signals are tied to $I(t)$ for ACF calculation). This changes the multi- τ estimator Eq. 9 to:

$$\hat{g}_{\text{sym,multi-}\tau}^{(\text{CCF})}(\tau_{s,p}) = \frac{G_{\tau_{s,p}}}{2^s} \cdot \frac{T}{M_0^{(g)} \cdot M_{\tau_{s,p}}^{(l)}} \quad (11)$$

Here the normalization uses the monitors $M_0^{(g)}$ for the global and $M_0^{(l)}$ for the local signal ($M_{\tau_{s,p}}^{(l)} = M_0^{(g)} \cdot (T - \tau_{s,p})/T$). For autocorrelation (see above) only one monitor channel is needed, as $M_0 = M^{(g)} = M^{(l)}$.

IV. PERFORMANCE & IMPLEMENTATION DETAILS

The complete design is implemented in two Virtex-II Pro FPGAs (XC2VP40, Xilinx, <http://www.xilinx.com/>, San Jose, USA), on a LASP development board¹⁶. One FPGA is used for data acquisition and line reordering, while the other one is used to implement the correlators. The total resource consumption within the second FPGA is around 80%. Using this hardware platform, there are currently $P = 8$ channels within each of the $S = 14$ linear correlator blocks.

In Eq. 7 we showed that the complete multi- τ correlator can be executed within twice the time needed for the first linear correlator block, so we devote half of the execution time to the first linear correlator and the rest to the remaining blocks. Therefore a new input sample can be accepted only once every $2\Delta t_{\text{lin}}$. Here $\Delta t_{\text{lin}} = 2P + 3$ cycles is the time needed to process a new input sample I_n in the first linear correlator block. The 3

data word	usage
0...111	delay register $J_{s,p}^{(l)}$ and raw accumulator values $G_{\tau_{s,p}}$
112	status counter c
113...125	intermediate results for accumulated input signals $I_{s,n}$
126	global monitor channel $M_0^{(g)}$
127	local monitor channel $M_0^{(l)}$

TABLE III. Memory layout of a pixel context

additional cycles are used for data hand over to the next block.

The CorrPEs run with a clock frequency of 144 MHz, which corresponds to an execution time of $264\text{ ns} = 2\Delta t_{\text{lin}}$. This is the minimum timespan between two subsequent samples, if no pixel multiplexing is used. Hence, our current FPGA platform can calculate 32 different ACFs or CCFs with a minimum lag time of 264 ns, which is comparable to the 100 ns designs presented in Ref. 17 and more recently in Ref. 4. Trading time resolution for more correlation functions, we can process all 1024 pixels of the SPAD array at a frame rate of 100 kfps in real time.

To estimate the memory consumption of our design, we first look at the pixel context, which consists of 128 words of 64 bits each. TAB. III shows a detailed memory layout. Sixteen of the upper 32 bits of the raw accumulators store the current value of the delay registers $J_{s,p}^{(l)}$. The lower 32 bits contain the accumulator $G_{\tau_{s,p}}$. The monitor channels are 32 bits each. Data handover between consecutive blocks (accumulated local and global signals) is done via 16 bit-wide memory areas, which is sufficient for $S \leq 16$. Since in the later linear correlators the accumulated input signals $I_{s,n}$ are multiplied, the sums $G_{\tau_{s,p}}$ and also the $I_{s,n}$ can get relatively large and may not fit in the 32 bit memory locations available. However, for our FCS application we estimate that the word sizes used are sufficient, since the per-pixel input event rate is limited. For a deeper discussion of this topic, see Ref. 18.

For our 32×32 pixel SPAD array the pixel contexts are stored in $32 \cdot 32 \cdot 128 \cdot 64\text{ bits} = 512\text{ KBytes}$ of external SRAM. A second SRAM stores the 256 KBytes used for the FIFOs (2048 entries each) that hold the pixel data until they are processed.

The correlator design is implemented in VHDL (very high speed integrated circuit hardware description language). It can be configured via generics (a VHDL feature). Thus we can reuse our design for different needs (e.g. for different sensor sizes and frame rates). We expect our design to show a significant increase in performance when implemented on newer FPGA generations (e.g. Virtex 5 or Virtex 6 from Xilinx).

To ensure the functional correctness of the designed correlator, a simulation of the hardware was tested using random input data. The outcome was compared to the results of a software implementation of the multi- τ cor-

relator using the same data set. Both yielded exactly the same results. The comparison was also done using real experimental data from the SPAD array. Again, both results were identical.

To demonstrate the functionality of the whole system, we tested the design using the SPAD array to record an LED connected to a sine wave generator set to 2.5 kHz. FIG. 6 shows the results of this measurement. A fit to the data recovered the chosen frequency.

V. CONCLUSION

In this paper we presented the implementation of an FPGA-based multi- τ correlator design that can calculate 1024 correlation functions in real time at a minimum lag time of $10\text{ }\mu\text{s}$. To our knowledge this is the largest number of real-time multi- τ correlators implemented so far in a single device. The minimum lag time of $10\text{ }\mu\text{s}$ in our design is considerably longer than that of currently available hardware correlators (e.g. from ALV GmbH, Langen, Germany or correlator.com, Bridgewater, USA and Ref. 19). However, those are limited to at most 32 auto-correlators.

We use our design to correlate the output of a single-photon avalanche diode array used as image sensor. This combination will allow us to perform imaging fluorescence correlation spectroscopy at sufficiently fast time scales to resolve even the motion of small molecules in solution and living cells.

Our design is flexible, so beside estimating ACFs, temporal cross-correlation functions of different pixels^{20,21} or multiple colors²² (using spectrally resolved detectors) can also be calculated.

ACKNOWLEDGMENTS

The project was supported by a NUS-BW (National University of Singapore / Baden-Württemberg) joint grant to J.L., a doctoral fellowship of the Helmholtz International Graduate School for Cancer Research to J.B., a doctoral fellowship of the Heidelberg Graduate School of Mathematical and Computational Methods for the Sciences to J.W.K. We thank Xilinx, San Jose, USA for donating the FPGAs on the LASP development board. G.V. receives support by the German-Hungarian program for the exchange of researchers by the German Academic Exchange Service and the Hungarian Scholarship Board (MÖB-47-1/2010) OTKA K77600 and TAMOP 4.2.1/B-09/1/KONV-2010-007.

Appendix A: Relation Eq. 8

By definition, Eq. 8 can be rewritten in the following manner, with C_s containing the cycles in which the linear

correlator s is processed:

$$\begin{aligned} C_0 &= \{c = n \cdot 2^1 + 0 \mid c, n \in \mathbb{N}_0\} \\ C_1 &= \{c = n \cdot 2^2 + 3 \mid c, n \in \mathbb{N}_0\} \\ C_s &= \{c = n \cdot 2^{s+1} + 2^s - 3 \mid c, n \in \mathbb{N}_0\} \forall s \geq 2 \end{aligned}$$

It remains to show that $C_p \cap C_q \forall p, q \in \mathbb{N}_0, p \neq q$. Obviously $C_0 \cap C_s \forall s \in \mathbb{N}, k, l \in \mathbb{N}_0$:

$$\begin{aligned} &C_1 \cap C_2 = \emptyset \\ \Rightarrow &k \cdot 2^2 + 3 \neq l \cdot 2^3 + 1 \\ \Leftrightarrow &\underbrace{k}_{\in \mathbb{N}_0} \neq \underbrace{2l}_{\in \mathbb{N}_0} - \underbrace{\frac{1}{2}}_{\in \mathbb{Q}} \\ \Rightarrow &\checkmark \end{aligned}$$

Without limitations let $r < s; \forall r, s \in \mathbb{N}_0; r, s \geq 2; k, l \in \mathbb{N}_0$:

$$\begin{aligned} &C_r \cap C_s = \emptyset \\ \Rightarrow &k \cdot 2^{r+1} + 2^r - 3 \neq l \cdot 2^{s+1} + 2^s - 3 \\ \Leftrightarrow &\underbrace{k}_{\in \mathbb{N}_0} \neq \underbrace{l \cdot 2^{s-r} + 2^{s-r-1}}_{\in \mathbb{N}_0} - \underbrace{\frac{1}{2}}_{\in \mathbb{Q}} \\ \Rightarrow &\checkmark \end{aligned}$$

Appendix B: FCS Simulation & Fits

To test different types of autocorrelators, we used our FCS simulation program described in Ref. 15. It simulates the 3D trajectories

$$\vec{r}_i(t), t = 0 \dots T_{\text{sim}}, i = 1..K$$

of a set of K fluorescing particles performing a random walk in which each 1D step is drawn from a zero-centered Gaussian distribution with width

$$\sigma_{\text{jump}} = \sqrt{2D \cdot \Delta t_{\text{sim}}}.$$

Here D is the given diffusion coefficient and Δt_{sim} is the simulation time step. We use a Gaussian approximation for the focal illumination and detection volume:

$$h(x, y, z) = \exp \left(-2 \cdot \frac{x^2 + y^2}{w_{xy}^2} - 2 \cdot \frac{z^2}{\gamma^2 w_{xy}^2} \right),$$

where w_{xy} is the lateral width of the profile, $\gamma = w_z/w_{xy}$ is its aspect ratio and w_z is its length. The program then estimates the expected number of photons $\bar{N}_{\text{phot}}(t)$ detected during each simulation time step $[t, t + \Delta t_{\text{sim}}]$:

$$\bar{N}_{\text{phot}}(t) = \bar{N}_0 \cdot \Delta t_{\text{sim}} \cdot \sum_{i=1}^K q_f \cdot q_{\text{det}} \cdot h(\vec{r}_i(t))^2.$$

T_{sim}	duration of simulation	1 s
Δt_{sim}	simulation time step	1 μ s
K	number of walkers	1576
V_{sim}	simulation volume	524 μm^3
c	concentration of simulated particles	5 nM
D	diffusion coefficient	20 $\mu\text{m}^2/\text{s}$
\bar{N}_0	absorbed photons per molecule	$4.2 \cdot 10^6 \text{ s}^{-1}$
w_{xy}	lateral width of excitation profile	0.325 μm
q_f	fluorescence quantum yield	0.8
q_{det}	detection efficiency	0.5
γ	focal aspect ration	6
S	number of linear correlator blocks	20
P	number of channels per block	16
m	lag time ratio between blocks	2

TABLE IV. Simulation parameters for the FCS simulation yielding the results, displayed in FIG. 5.

Here \bar{N}_0 is the maximum number of detected photons per fluorophore and time step, while q_f and q_{det} are the quantum efficiencies of fluorescence and detection. To account for the counting statistics in the SPADs, the number of photons $N_{\text{phot}}(t)$ actually detected during a simulation time step is calculated by drawing a random number from a Poissonian distribution with average $\bar{N}_{\text{phot}}(t)$.

The time series created by the simulation is then fed into the three different autocorrelator software implementations: (a) direct estimation, (b) multi- τ with one monitor channel and (c) multi- τ with multiple monitor channels. The latter two (b, c) simulate the complete structure of a multi- τ correlator. In correlator (b) we use one single monitor channel, which counts the incoming photons only and then uses Eq. 10 for the normalization. In the correlator (c) we use one monitor channel per correlator lag which counts the photons actually processed by each lag.

The simulation parameters for the data in FIG. 5 are summarized in TAB. IV. The average photon count rate in the simulations was $\langle I(t) \rangle \approx 157.4 \text{ kHz}$ which is – due to the chosen detection efficiency $q_{\text{det}} = 0.5$ – about a factor of 5 higher than what would be expected from a standard confocal FCS setup. This way the signal to noise ratio is high enough to visualize easily artifacts introduced by the correlator implementation. FIG. 7 shows an example of a time trace $N_{\text{phot}}(t)/\Delta t_{\text{sim}}$ generated by our simulation.

The resulting ACFs were fitted to a standard 3D FCS model function³:

$$g^{(\text{fit})}(\tau) = \frac{1}{N} \cdot \left(1 + \frac{\tau}{\tau_D} \right)^{-1} \cdot \left(1 + \frac{\tau}{\gamma^2 \tau_D} \right)^{-1/2}$$

with the average particle number N in the effective focal volume $V_{\text{eff}} = \pi^{3/2} \cdot \gamma \cdot w_{xy}^3$ and the diffusion decay time $\tau_D = w_{xy}^2/4D$. The fits were performed using a Levenberg-Marquardt least squares fitting routine.

REFERENCES & FOOTNOTES

- ¹D. Magde, E. L. Elson, and W. W. Webb, *Biopolymers* **13**, 1 (1974).
- ²D. Magde, E. L. Elson, and W. W. Webb, *Biopolymers* **13**, 29 (1974).
- ³O. Krichevsky and G. Bonnet, *Reports on Progress in Physics* **65**, 251 (2002).
- ⁴G. Mocsár, B. Kreith, J. Buchholz, J. W. Krieger, J. Langowski, and G. Vámosi, *Review of Scientific Instruments* ?, ? (2011).
- ⁵B. Kannan, L. Guo, T. Sudhaharan, S. Ahmed, I. Maruyama, and T. Wohland, *Analytical Chemistry* **79**, 4463 (2007).
- ⁶N. Dross, C. Spriet, M. Zwerger, G. Müller, W. Waldeck, and J. Langowski, *PLoS ONE* **4**, e5041 (2009).
- ⁷T. Wohland, X. Shi, J. Sankaran, and E. H. K. Stelzer, *Optics Express* **10**, 10627 (2010).
- ⁸The single SPADs of Radhard2 have a photon detection probability of around 35% at a wavelength of 525 nm. The fill factor of the array is around 1.5%.
- ⁹L. Carrara, C. Niclass, N. Scheidegger, H. Shea, and E. Charbon, in *ISSCC, IEEE International Solid-State Circuits Conference* (2009) pp. 40–41.
- ¹⁰K. Schätzel, *Quantum Opt.* **2**, 287 (1990).
- ¹¹K. Schätzel, Institute of Physics Conference Series **77**, 175 (1985).
- ¹²Z. Kojro, A. Riede, M. Schubert, and W. Grill, *Review of Scientific Instruments* **70**, 4487 (1999).
- ¹³J. Sankaran, X. Shi, L. Ho, E. Stelzer, and T. Wohland, *Optics Express* **18**, 25468 (2010).
- ¹⁴The diffusion coefficient was $D = 20 \mu\text{m}^2/\text{s}$ (corresponding to an intermediately sized protein in water), the simulation timestep of the random walk, as well as the minimum lag time were $\Delta t_{\text{sim}} = \tau_{\text{min}} = 1 \mu\text{s}$. There were around 1.2 particles in the effective measurement volume $V_{\text{eff}} \approx 0.4 \mu\text{m}^3$ on average.
- ¹⁵T. Wocjan, J. Krieger, O. Krichevsky, and J. Langowski, *Phys. Chem. Chem. Phys.* **11**, 10671 (2009).
- ¹⁶C. Niclass, C. Favi, T. Kluter, M. Gersbach, and E. Charbon, in *ISSCC, IEEE International Solid-State Circuits Conference* (IEEE, 2008) pp. 44–594.
- ¹⁷C. Jakob, A. Schwarzbacher, B. Hoppe, and R. Peters, in *Irish Signals and Systems Conference, 2006. IET* (IET, 2006) pp. 99–103.
- ¹⁸B. Hoppe, H. Meuth, M. Engels, and R. Peters, in *IEEE Proceedings Circuits, Devices and Systems*, Vol. 148 (IET, 2001) pp. 267–271.
- ¹⁹D. Magatti and F. Ferri, *Review of Scientific Instruments* **74**, 1135 (2003).
- ²⁰T. Dertinger, V. Pacheco, der Hocht Iris von, R. Hartmann, I. Gregor, and J. Enderlein, *ChemPhysChem* **8**, 433 (2007).
- ²¹R. A. Colyer, G. Scalia, I. Rech, A. Gulinatti, M. Ghioni, S. Cova, S. Weiss, and X. Michalet, *Biomedical Optics Express* **1**, 1408 (2010).
- ²²F. Bestvater, Z. Seghiri, M. S. Kang, N. Gröner, J. Y. Lee, I. Kang-Bin, and M. Wachsmuth, *OPTICS EXPRESS* **18**, 23818 (2010).

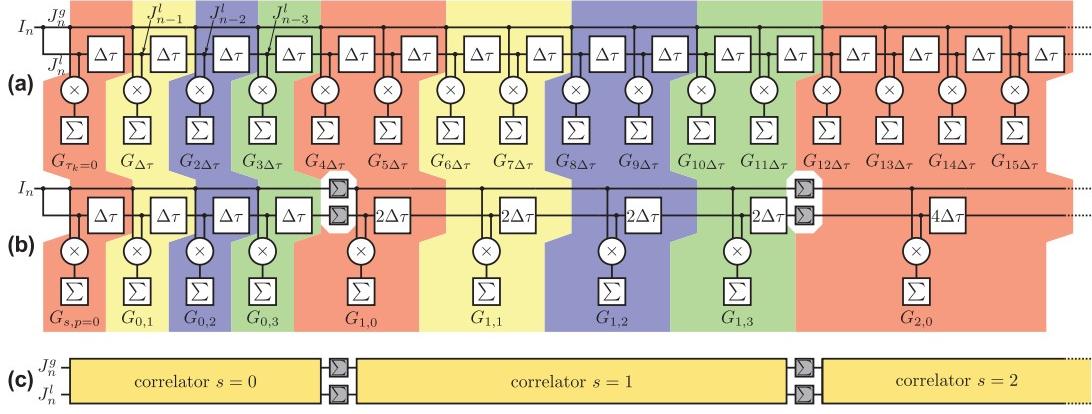


FIG. 1. Hardware design of a linear correlator (a) in comparison with a multi- τ correlator (b). Panel (c) shows a schematic view of the multi- τ correlator, where the linear correlator building blocks are summarized by a single block. Corresponding channels are grouped with the same color. Global/undelayed (g) and local/delayed (l) inputs are located on the left. For autocorrelation, the global and local signal inputs to the 0-th block ($J_n^{(g)} = J_n^{(l)} = I_n$) are identical. The $\Delta\tau$ -blocks represent delay elements, the \otimes -blocks multipliers and the Σ -blocks accumulators.

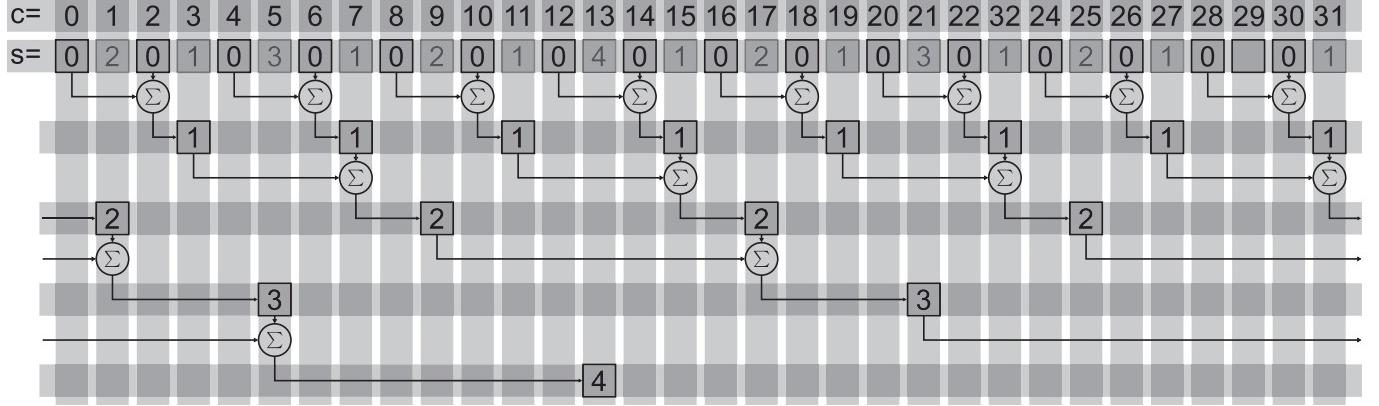


FIG. 2. Block scheduling algorithm. The counter c in the first row defines the current cycle. The second row shows the block s that is processed in the corresponding cycle. Further below the order of processing is shown: When a block s has been processed twice, the following block, $s + 1$, can process the sum of the two preceding s -blocks.

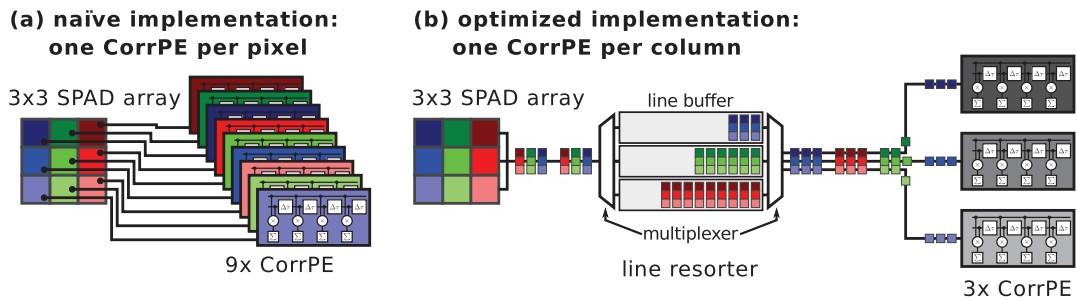


FIG. 3. Comparison of a naïve implementation (a: one correlator per pixel) and an optimized implementation (b: reuse CorrPEs for several pixels) of a multi-pixel multi- τ correlator for a 3×3 SPAD array.

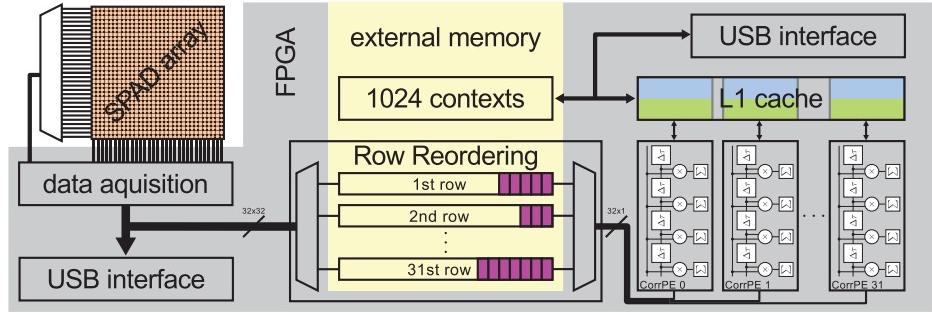


FIG. 4. System layout and data path - from data acquisition to correlation. One USB interface is used to stream the raw images, the other for streaming (intermediate) results. The first level cache (L1, double buffered) is used to hold the context of the currently processed pixel. FIFOs for row resorting and for context storage use external memory.

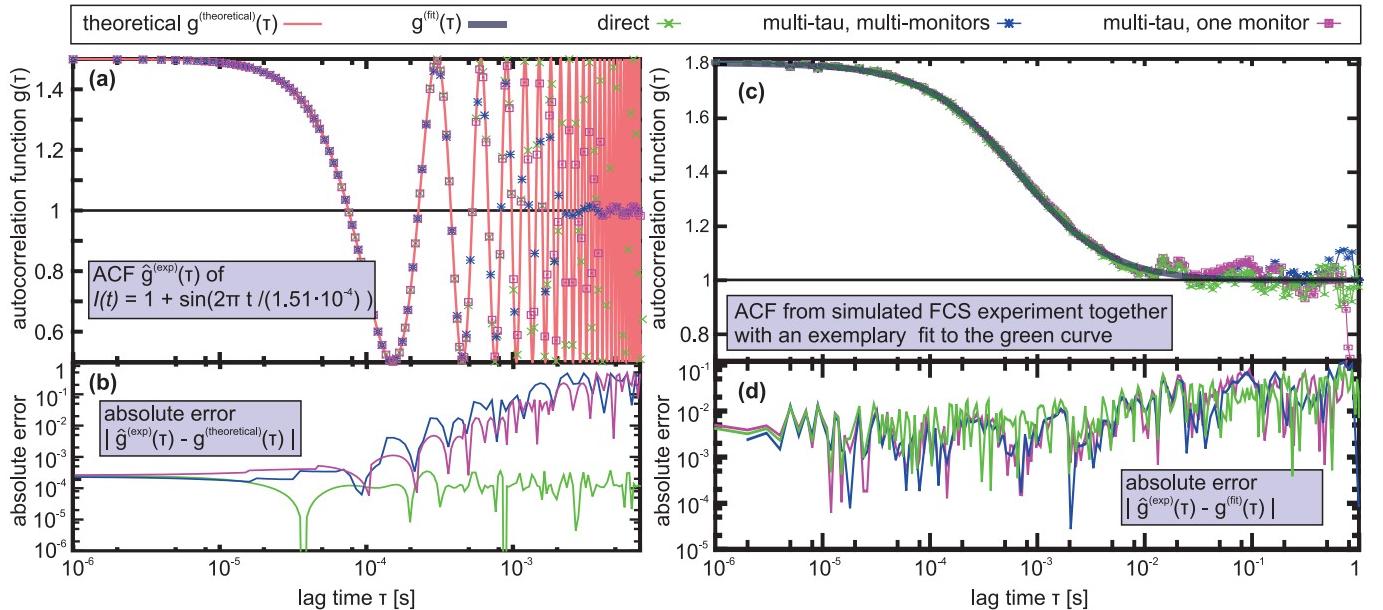


FIG. 5. Simulation results for different implementations of multi- τ correlators: The left panels (a,b) show simulations for a sine wave input signal $I(t) = 1 + \sin(2\pi t/(0.151 \text{ ms}))$. The simulations on the right (c,d) were created using an FCS simulation. The top graphs (a,c) are estimates of the autocorrelation function using direct correlation from Eq. 3 (green), a multi- τ correlator with one monitor channel per lag time (blue) and our estimated normalization from Eq. 10 (magenta). Graph (a) also shows the theoretical ACF $g^{(\text{theoretical})}(\tau) = 1 + \cos(2\pi\tau/(0.151 \text{ ms}))$ for the sine signal (light red). The lower graphs (b,d) show the absolute deviation of the estimates from $g^{(\text{theoretical})}(\tau)$ (b) and from a fit to the curves (d). The parameters resulting from the fit in (c) are the same within $< 2.5\%$ for all three estimates.

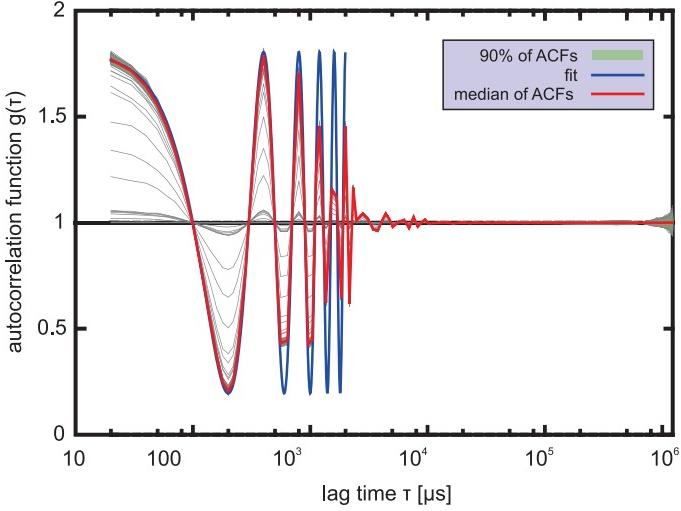


FIG. 6. Distribution of 992 (gray) ACFs taken by our sensor (first 31 columns only), exposed to a 630 nm LED sine-modulated with a frequency of 2.5 kHz. The correlator was running for 1.2 s (131072 samples at $\tau_{\min} = 10 \mu\text{s}$). The gray curves with significantly lower amplitude are due to hot pixels of the SPAD array; since these SPADs fire randomly, the corresponding correlation amplitude is decreased. The median, which tends to be less sensitive towards outliers than the mean, is shown in red. The theoretical model $g^{(\text{theoretical})}(\tau) = 1 + A \cdot \cos(2\pi f \cdot \tau)$ is fitted against the median in the interval $[10 \mu\text{s}, 1 \text{ ms}]$, and is shown until $\tau = 2 \text{ ms}$. The fit yields a frequency of $(2502 \pm 4) \text{ Hz}$. The fanning out of the curve at high values of τ is due to unfilled correlator channels. Compare also FIG. 5(a).

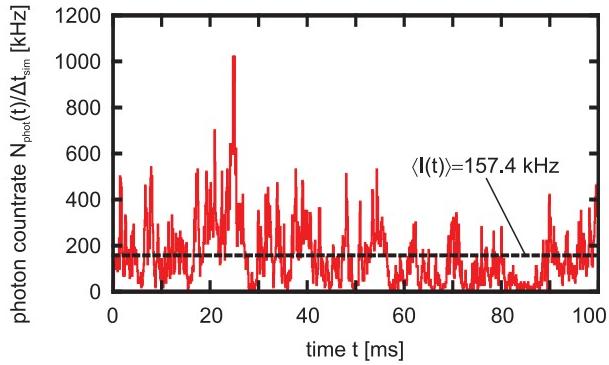


FIG. 7. Typical photon count time trace $N_{\text{phot}}(t)/\Delta t_{\text{sim}}$ generated by our simulation